General ML

Types of error:
(1) modeling error caused by overly simple model. It happens when both training and validation acc are low.
(2) estimation error caused by insufficient training data. It happens when training accuracy is very high but testing accuracy is low.
(3) optimization error caused by imperfect optimization, which makes the model not reach the full convergence.
(4) Bayes error is irreducible. It can be caused by inexpressive features or contradicting samples (same X but different y)

Decision Tree

Entropy: $H = -p_+ log_2 p_+ - p_- log_2 p_-$
Information gain: $H(r) - \sum_{i=1}^{k} p_i H(c_i)$, where r is root, c are chilren, and p_i is the probability of samples going to the i^{th} nodes
Mutual Information: symmetric and non-negative (can't decrease uncertainty by knowing more)
Greedy Algorithm for building decision tree
(1) Pick the attribute that best separate the samples (measured in purity or entroy)
(2) Go down the tree and continue to pick the node the best separate the samples, repeat this until some conditions are met (pure nodes)
Properties: (1) easy to understand (2) deal with both discrete and continuous without the need for normalization or preprocessing (3) highly flexible hypothesis space, which can lead to overfitting (4) non-linear classifier
Avoid overfitting: (1) Post pruning: remove nodes that have little impact on validation accuracy (using holdout set) (2) Prepruning (early stop) by stop growing the tree when the validation accuracy increase is insignificant. This can cause underfitting because subsequent splits might reduce error significantly.
Multi-nomial features: might cause data fragmentation (too little data might fall into each subtree) → overfitting.
Notes: (1) both continuous features and binary features can appear many times in a tree, but continuous features can appear multiple times in a single path whereas binary features can only appear once on each path (2) for regression tree, uncertainty of a branch can be measured by calculating sum squared error (variance)

Clustering

Hierarchical Clustering: there are 2 approaches, bottom up (agglomerative) and top down (divisive)
Hierarchical Agglomerative Clustering: starts with each sample in a separate cluster, then merge the pair of clusters that have the smallest "distance"until there is only one cluster left.
(1) Single Link: distance between the nearest pair of points across two clusters
(2) Complete Link: distance between the furthest pair of points across two clusters
(3) Average link: average distance of all cross cluster pairs
(4) Centroid: distance between the means of the two clusters
Single Link vs. Complete Link (1) Single link can produce straggling clusters (chains of clusters) whereas complete link creates clusters with roughly equal size → creates more useful organization of data (2) complete link might pay too much attention to outliers
Interpret Dendrogram: (1) a horizontal cut creates a partition (2) large gaps indicate good cutting points
Flat Clustering: K-means and Mixture of Gaussian
Q1: complete link and average link capture hypersphere of similar sides, while single link yields arbitrary shape. Kmeans assumes assume hypersphere shape while mixture of Gaussian assume Gaussian clusters.

Dimensionality Reduction

Supervised Dimension Reduction: Linear Disciminant Analysis (LDA): (1) find a projection to maximize the distances between the means (distances between clusters) and (2) minimize the "scatters"(distances between points within one cluster) (3) can be viewed as a generative classifier p(x—y): Gaussian with distinct μ for each class but a shared σ
Unsupervised Dimension Reduction: (1) *Projection*: Principal Components (PCA), and (2) *Manifold Learning*: t-Distributed Stochastic Neighbor Embedding (t-SNE) and ISOMAP
Motivation: (1) help visualize and interpret high dimensional data (2) remove noise/irrelevant features → reduce computational cost, overfitting, and increase efficiency

K-means

Assume that all clusters have the same shape (same distribution)???
Objective: minimize the distances from the points to the centroids of their clusters
Algorithm:
(1) Randomly pick k centroids
(2) Assign the samples to the closest clusters (the closest centroids)
(3) Update the centroids to reduce the cost (inertia), if there is at least a centroid update, go back to step (2)
Metrics: (1) inertia: mean squared distance between each sample and its closest centroid
Properties: (1) each iteration is guaranteed to decrease the objective and there are finite classes → guarantee to converge but only to local minimum (2) highly sensitive to initial seeds → should try multiple random initializations and pick the one with the least sum of squared loss (3) outliers can cause problems such as making singular clusters or bias the centroid estimation
k-medoids++: (1)(+) use cluster medoids (the centered sample) instead of centers (imaginative), similar to median vs. mean → more robust to outliers (2)(-) more expensive to compute (3)(-) need to predefine k
kmeans++: choose initial centers to be far apart so that they are more likely to be in different clusters.
Notes: Kmeans is a special case of mixture of Gaussian

PCA

Idea: (1) identifies the hyperplane that lies "closest"to the data, (2) projects the data onto it.
Objective: (1) maximizes variance (most likely lose less information) or (2) minimizes the mean squared distance between the original dataset and its projection
Notes: (1) PCA assumes the dataset is centered around the origin, (2) choose the number of dimensions that add up to some percentage of the variance (often at least 95%) of the original dataset. (3) linear
Q1: top component directions do not guarantee to provide the best separation for classification. Because these components are chosen to maximize the projected variance and some important features for classification may have low variance, and therefore are diminished

Expectation Maximization

(1) start with an initial guess of the model parameters (2) given the current parameters λ_t, compute the expectation for the hidden data (3) re-estimate the parameters λ_{t+1}

Algorithm: repeat until convergence (θ^n: current parameters)

(1) E-step: For each data point i, compute posterior of z_i: $q_i(z_i) = p(z_i|x_i;\theta^n)$

(2) M-step: maximize the expected log-likelihood

$$\theta^{n+1} = arg\max_{\theta} \sum_i \sum_{z_i} q_i(z_i) logp(x_i, z_i; \theta)$$

Example: 100 data points generated from a mixture of two uniform distributions: uniform(0,1) and uniform(0, 6). Let p and 1-p denote the prior probability for uniform(0,1) and uniform(0,6) respectively. Let y=1 denote uniform(0,1) and y=0 denote uniform(0,6). Assume that current estimate is p=0.4

$P(y = 1|x = 0.5) = \frac{P(x=0.5|y=1)P(y=1)}{P(x=0.5)} = \frac{1\times0.4}{P(x=0.5|y=0)+P(x=0.5|y=1)} \frac{0.4}{1\times0.4+\frac{1}{6}\times0.6} = \frac{4}{5}$

Maximization step: new estimate $p = \frac{P(y=1|x\leq1)\times60+P(y=1|x>1)\times40}{100} = \frac{0.8\times60+0\times40}{100}$

Notes (1) used to handle hidden (missing) data (2) guarantee to increase the likelihood objective every iteration → guarantee to converge but slow in practice, stop early can be used with a threshold of log-likelihood change (3) converges to a local optimum → use multiple starts then choose the solution with the highest marginal likelihood

Buffer

1
2
3
4
5
6
7
8

ISOMAP

Observation: data often lies on a near or nonlinear low-dimensional curve, which is called manifolds

Notes: (1) nonlinear (2) preserve the global, nonlinear geometry of the data by preserving the geodesic distances (3) sensitive to the construction of graph, too few neighbors → the graph might not have sufficient edges, too many neighbors → might connect points that shouldn't be (4) A general strategy is to try different parameters for graph construction and visualize the results

Q1: if the high dimensional data is very sparse, the shortest path approximation to the geodesic distance will be poor. Then large values of epsilon have to be used to build a connected graph, which might make the connections jump accross regions, leading to an invalid approximation of the geodesic distance.

t-SNE

Idea: converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

Properties: (1) preserve local structure (2)

Optimization: There are 5 parameters that control the optimization of t-SNE:

(1) perplexity: $k = 2^{(S)}$ where S is the Shannon entropy of the conditional probability distribution. k is the number of nearest neighbors t-SNE considers when generating the conditional probabilities. Larger k → more nearest neighbors → less sensitive to small structures.

(2) early exaggeration factor (3) learning rate (4) maximum number of iterations (5) angle

Cons: (1) computationally expensive (2) stochastic, therefore different initializations yield different results (3) global structure it not explicitly preserved (4) can't be applied to new data points

Notes: (1) has non-convex cost function, hence different initializations might result in different results

Q1: when perplexity is set too small, then the effective neighborhood will be very small. Recall that t-SNE solution only care about these pairs that have high p_{ij} values (neighbors). This will mean that only the closest pair of points will matter in forming the embedding, leading to a solution that has lots of small groups randomly scattered around.

Q2: If the perplexity value is too large then entropy and size of the neighborhood will be so large that all

Neural Network

Activation Functions: (1) Sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, (2) Tanh function $tanh(x) = 2\sigma(2x)-1$, (3) Rectified Linear Unit (ReLu) function $f(x) = max(0, x)$ (4) perceptron f(x) = 1 if $w.x + b > 0$, otherwise f(x) = 0.
Representation Power:
Universial Function Approximator (optional)
Backpropagation Training: (1) aims to find weights that minimize some loss function (2) applies chain rule for gradient, starting from output back to the first layer. $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$
Correction signal delta to weight k for sample i: $\frac{\partial J^i}{\partial W_k} = \delta_k^i A^i$, where $\delta_k^i = (\hat{y}^i - y^i)\hat{y}^i(1-\hat{y}^i)$ and A is the output of the hidden layer.
Impact of ReLu: (1) reduces the issue of vanishing gradient (happens when certain activation functions like sigmoid squishes a large input space into a small space from 0 to 1, which makes the derivative become small), (2) introduces sparsity in the hidden layer activations, which makes the model less prone to overfitting
Training:
(1) Batch learning sums up the gradient for all of examples and take a combined gradient step
(2) Online learning takes a gradient step for each example
(3) Momentum learning combines the current gradient with the previous update direction to ensure smoother convergence
Remarks on Training: (1) not guarantee to converge, may oscillate or reach local minima → try different initializations (2) requires lots of data (3) might be overtrained, requires early stopping using holdout validation set to detect generalization error degradation (4) two few hidden units → underfitting, too many → overfitting. Cross-validation can be used to search for a good number. Weight decay (multiply weights with a number between 0 and 1) can help mitigate overfitting, similar to reguralization. (5) appropriate input/output encoding is crucial (6) use a good activation function is important
Proper initialization: (1) start with simple models by keeping all weights near zero to avoid getting into linear region of some activation function like sigmoid/tanh → avoid saturation and too small gradient (2) and different from one another to learn from different "angles"; if all weights are initialized to the same value, each hidden unit will receive the same signal and the weights will remain same due to same updates
Q1: Relu shuts off negative input → makes activation

Mixture of Gaussian

*

Ensemble Methods

Bagging: derive many different training sets by sampling the original training set, and use them to train multiple models independently. The resulting models are then aggregated by voting mechanism (1) does not impact bias, but reduces variance (2) tends to work well with base classifiers that have low bias and high variance such as decision trees, neural networks (3) stable (low variance) classifiers (perceptron, logistic regression) do not draw much benefit from bagging (4) the ensemble tends to be smoother than individual models (5) robust to noise and outliers
Random Forest: (1) an extension of bagging, combine many high-variance and low-bias trees to create an ensemble of low variance (high confidence) by using data sample bagging and feature bagging (2) robust to noise and outliers (3) efficient for large dataset (4) can be used to estimate variable importance
Boosting: (1) contrast to bagging, instead of let the models learn independently, it forces the models to focus on different input space, hence make different errors. In particular, in each iteration, focus more on errors from previous classifiers and less on examples that are correct (weighted training). Each classifier is measured in terms of accuracy; more accurate classifier gets more weight in final voting. (2) often (not always) robust to overfitting (3) test error continues to decrease even after training error goes to zero (+4) can identify outliers since it focuses on samples that are hard to categorize (-5) too many outliers can degrade classification performance dramatically and increase time to convergence
Bias Variance Decomposition loss can be decomposed into three parts: bias, variance, and noise (irreducible error).

$$Err(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_e^2$$

(1) Bias corresponds to modeling error. Models with high bias have too small hypothesis space, which can be fixed by using more complex hypothesis space.
(2) Variance corresponds to estimation error and possibly optimization error. It happens when the hypothesis space is very complex and the individual learned hypothesis has highly varying performance on different training set. This can be reduced by increasing the training set size
(3) Noise mainly corresponds to the bayes error, which is irreducible regardless of training data size and model complexity
Q1: adding more classifiers in a bagging ensemble does